DIN OF AN "IQUITY": ANALYSIS AND SYNTHESIS OF ENVIRONMENTAL SOUNDS

Perry R. Cook (with lots of help)[#]
Princeton University Soundlab
Department of Computer Science (also Music)
Princeton, NJ, USA
prc@princeton.edu

ABSTRACT

This paper describes a series of related research and software projects in the analysis and synthesis of stochastic sounds in general, and more specifically, applications in the synthesis of environmental sounds. Analysis and synthesis of sounds as varied as a maraca (many beans bouncing around in a gourd), to groups of noise-making animals and insects, to human applause will be covered. Specific open-source project software will be described, such as the "Shakers" and "Flox" classes in the Synthesis ToolKit in C++ (STK), GaitLab (analysis and synthesis of walking sounds), ClapLab and ClaPD (synthesis of applause), TAPESTREA (Techniques and Paradigms for Expressive Synthesis and Transformation of Environmental Audio), and the new audio programming language ChucK.

[Keywords: Environmental Sound, Background Sound, Din]

1. INTRODUCTION

To begin, the author should explain the liberty taken in coining a new word, "iquity" in the title (a pun on "den of iniquity, often used to describe houses of ill repute, opium dens, hashish parlors etc.). "Iquity" comes from the word "iniquity," meaning injustice or wickedness, whose etymology is from "in" meaning not, and the Latin "aequus" meaning equal.

"Din" is defined as a collection of discordant sounds or constant noise. We define it as background sound, or that which is left after one accounts for and removes all foreground sounds. Examples of foreground sounds might include the person close to us at the cocktail party talking directly to us, or a horn honking on a busy street. The din in these cases would be the mixture of other conversations (minus our conversation) or the noise of the street minus the horn honk. So "Din of An Iquity" refers to our attempts to do justice to the background sounds, or to do as good a job as possible (or computationally affordable), to give the *impression* of the din we attempt to model (perceptual equality).

A few researchers have investigated the modeling of continuous background sound [1][2], often referring to it as audio "texture." Indeed there is a large literature in the graphics community on visual texture modeling and synthesis, and in the haptics (combined senses of touch) community on modeling and synthesizing the "feel" of objects, including their texture, using computer-driven motors and vibrators.

The projects described here assume that the acoustic source of many environmental sounds is an ensemble of individual sound-producing objects or entities, joining to make a perceptual whole. So a "bunch of hand claps" might be called applause, or a collection of small metal cymbals attached to a shaken ring might be called a tambourine. A gaggle of geese has a sound unique from a swarm of locusts, or a collection of many different conversations at a cocktail party. We do not endeavor to model all of these in this paper, but do attack a number of them.

2. RANDOM PHYSICAL EVENT MODELING

In 1995 the author launched into a new research agenda aimed at physical modeling of the most varied single section of any orchestra, the so-called "percussion section," which actually includes pretty much anything that isn't a bowed-string or wind instrument. Drums of all kinds, mallet percussion (marimba, xylophone, glockenspiel, vibraphone, orchestral chimes), claves, castanets, shakers (maraca, tambourine, sleighbells, sekere), scrapers and ratchets (guiro, ratchet), brake drums and other found or manufactured metal/wood objects, and even the celeste (a keyboard-controlled set of orchestra bells) and piano often are counted among the percussion instrument "family."

Of most interest to the author were the shakers, scrapers, and ratchets; the noisy things that have a specific character, yet when a single sample is played back over and again it becomes perceptually obvious that it is a single sample. After doing a series of exhaustive simulations where all particles, (beans in a virtual maraca shell) were modeled in 3D, some observations about the physical acoustical system and the statistics of collisions were made that yielded a great simplification in the computational n-body algorithm. These observations were that:

1. Once excited (by shaking the maraca), the total kinetic energy in the system decays exponentially. Thus the radiated sound energy also decays exponentially.

2. Collisions between particles inside the outer shell do not cause sound to be radiated; only collisions of particles with the shell itself cause it to be excited.

3. The shell radiates the sound, while performing resonant filtering on the bean/shell collision impulses, and the characteristics of the shell filter are relatively constant.

4. The amount of excitation of the shell is proportional to the cosine of the angle between the incident particle and the shell normal, which is roughly random given:

5. The likelihood of sound-producing collisions follows roughly a Poisson distribution, as does the incident angle of the particle colliding.

These observations led to the PhISEM (Physically Inspired Stochastic Event Modeling) algorithm [3][4]. As a simple

example, here is the C code required to compute a simple maraca sound:

```
// ANSI C Code to Calculate Single Sample of Maraca Algorithm
#define SOUND_DECAY 0.95
#define SYSTEM_DECAY 0.999
shakeEnergy*=SYSTEM_DECAY;  // Exponential system decay
if (random(1024) < num_beans)       // If collision
   sndLevel += gain * shakeEnergy;  // add energy to sound
input = sndLevel * noise_tick();    // Actual sound is random
sndLevel *= SOUND_DECAY;            // Exponential Sound decay
input -= output[0]*coeffs[0];       // Do simple
input -= output[1]*coeffs[1];       //      system resonance
output[1] = output[0];              //         filter
output[0] = input;                  //              calculations
```

Looking around for other sound-producing systems which could be modeled by this algorithm (or simple extensions to it) yielded quite a large list including many of the orchestral percussion, but also many non-musical sounds ranging from ice cubes in an empty glass, to wind chimes, to leaves crunching under feet while walking [5][6]. These and others were implemented in the Shakers.cpp class of the open-source Synthesis ToolKit in C++ (STK)[7][8]. A total of five filters are available to implement resonances of the system being modeled, and algorithmic rules control how these filters are used depending on the system. Figure 1 shows the PhISEM model block diagram.

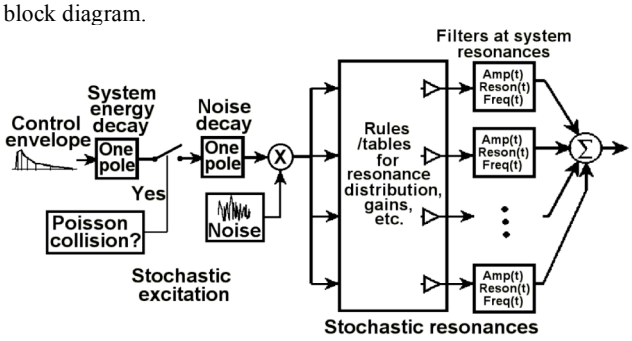

Figure 1. *PhISEM synthesis block diagram.*

The PhISEM algorithm has been used for a number of psychoacoustic experiments [9][10] as well as the synthesis of sound effects.

3. GAITLAB: MODELING OF WALKING SOUNDS

The observation that the "texture" underfoot while walking imparts a different character to the sound produced (walking on gravel, slogging through mud, crunching through snow, leaves, sticks, etc.) gave rise to the GaitLab project [11]. In this, a model of the pseudo-periodicity (and random variations) of footfalls from walking sounds was developed, and used to drive the PhISEM model, or random overlap-add playback of segments of the original sound. Figure 2 shows the GaitLab analysis/synthesis system block diagram. Figure 3 shows a simple GaitLab graphical user interface, with various controls for left/right symmetry randomization, etc.

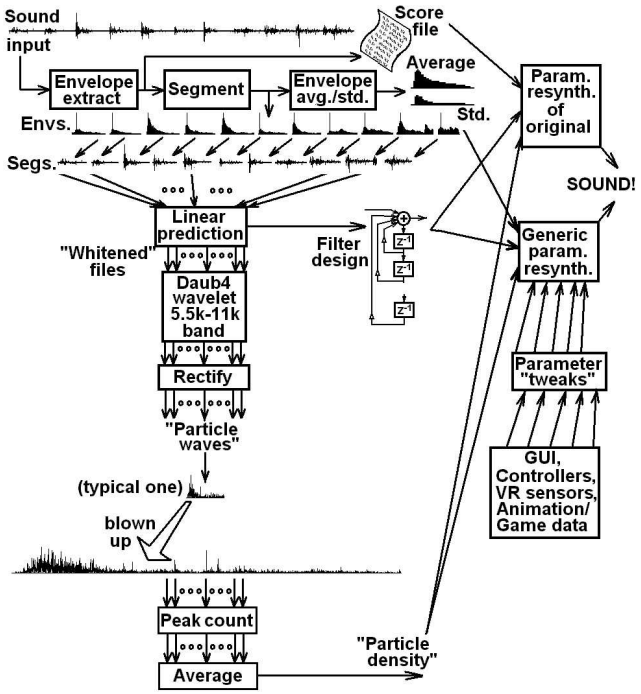

Figure 2. *"GaitLab" architecture. Sound is first segmented, parameters are extracted and parameterized, then synthesis is performed using either randomized segments of the original sound, or a parametrically driven PhISEM algorithm.*

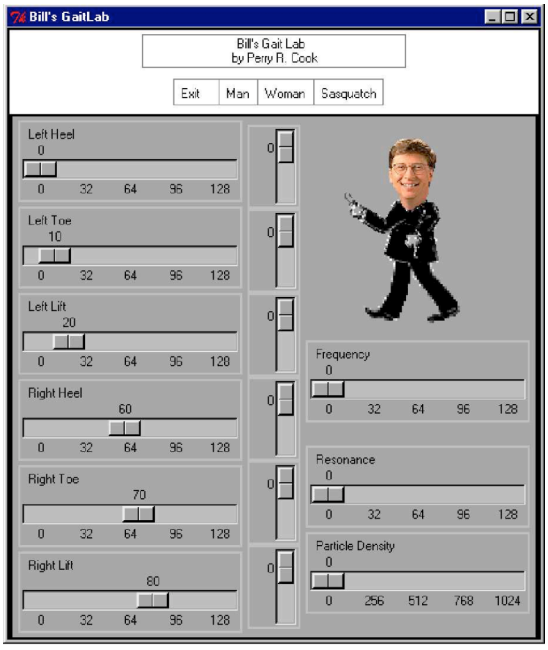

Figure 3. *Simple GaitLab synthesis control GUI.*

Figure 4 shows the PhOLeyMat from the PhOLISE (Physically Oriented Library of Interactive Sound Effects) project, in which force sensors beneath 9 different tiles are used to drive 9 differently calibrated GaitLab walking sound textures including grass, wood, coarse gravel, fine gravel, tile, and carpet.

Figure 4. GaitLab's "PhOLIEMat" PhISEM controller.

4. FLOX: SYNTHESIS OF SONIC "HORDES"

As previously mentioned, many environmental sounds are composed of multiple sound sources, acting independently, adding together to create the background din. Sometimes the sources are all of the same type, as is the case in applause, a flock of birds chirping, a forest full of crickets, and many other such collections. The Flox.cpp class, implemented in STK, allows control of from 0 to N sound producing objects. The maximum value of N is set when a new Flox instance is created. The sound producing objects can be shakers, clappers, crickets, frogs, etc. They can be short sound clips, or even musical models of plucked strings or marimbas. The Flox object controls the placement in the stereo field, triggering, resonant frequencies, etc. with control over randomization of each parameter.

4.1. Synthesis of Clapping and Applause

The Flox class in STK was created for synthesis of applause as an ensemble of analyzed individual clappers. The author found another group in Finland researching the problem [12], and [13] resulted as a joint publication on the similar (yet different) approaches to the topic. First data was collected from human hand clappers. A simplified (no need to estimate # particles) GaitLab (Figure 2) architecture worked well with only minor adjustments for segmenting, analyzing, and extracting parameters from the clapping sounds. Table 1 shows the mean and standard deviations of the period T (duration between claps) and center resonant frequency F1 for four male and four female clappers.

Subject	Mean T (s)	STD	Mean F1	STD
M1	.256	.0093	1203Hz	278
M2	.327	.0083	435	40
M3	.276	.0128	3863	1009
M4	.265	.0060	1193	243
M5	.238	.0061	1519	219
M6	.284	.0077	2243	775
M7	.298	.0100	1515	239
M8	.285	.0016	1928	764

Table 1: *Handclap statistics for four males and four females.*

Figure 5 shows the spectrum of a single handclap, superimposed with the spectrum of the impulse response of a low order (two pole) resonant filter designed by least-squares fit using Linear Predictive Coding (LPC).

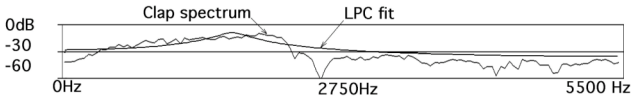

Figure 5. LPC fit to spectrum of single handclap.

The synthesis architecture of Figure 1 works well for clap synthesis when modified by replacing the Poisson probability calculation (the "// if collision" line in the C Code example above) with a periodicity calculation, with randomness to model the standard deviation in period. Again, a low order resonant filter works well for clapping.

As we know, audiences don't behave entirely as autonomous clappers, sometimes synchronizing, then falling out of phase, then back again, sometimes speeding up as well. Figure 6 shows the interface for ClapLab, which includes controls for mean and standard deviation (randomness) of center frequency, period (tempo), and "affinity" (the tendency of the clappers to clap in unison, with 0 causing completely random applause and 128 meaning perfectly synchronized applause. The #Objects slider selects individual clappers from the human subject data for numbers 1-8, and adds more clappers with random parameters for numbers 9-128, resulting in a maximum of 129 total clappers.

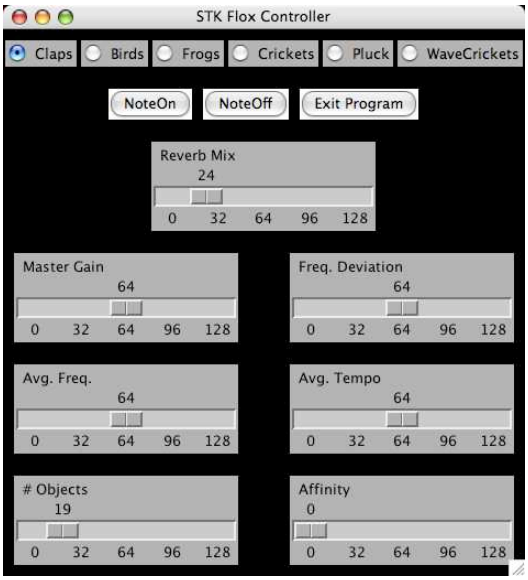

Figure 5. ClapLab graphical user interface.

4.2. Synthesis of Other Hordes, Flocks, Swarms, etc.

As mentioned previously, the STK Flox class can also be used to control other quasi-homogeneous noisemaking ensembles, such as birds, frogs, crickets, or even musical instrument sounds and models. Figure 5 above shows such selection buttons, and Figure 6 shows a display written in Open GL for displaying the events. Each "character" appears when their sound is triggered, then rapidly fades away in the graphical display.

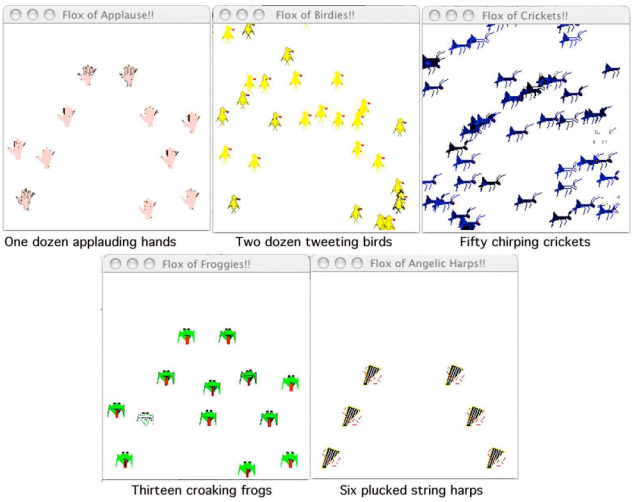

Figure 6. Flox GL display of synthesized noisemakers.

5. TAPESTREA

TAPESTREA is a technique and system for "re-composing" recorded sounds by separating them into unique components and weaving these components into sonic tapestries. The technique and system is applicable to sound-design [14], interactive sound environments [15], and musique concr`ete or acousmatic music composition [16]. The TAPESTREA analysis screen provides a GUI for interactively separating sound scenes into deterministic (sinusoidal) components [17], transients [18], and the remaining stochastic background sound (our definition of "din"). Figure 7 shows the overall system architecture of TAPESTREA.

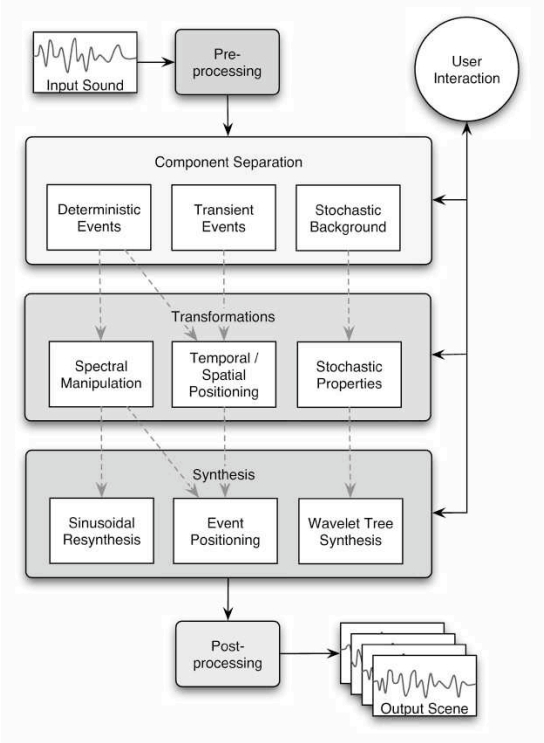

Figure 7. Architectural pipeline of TAPESTREA.

Figure 8 shows the sinusoidal/stochastic analysis screen with interactive waveform (time segment selectable) and spectral (an arbitrary rectangle can be selected in the spectrogram) displays, and controls for extraction parameters.

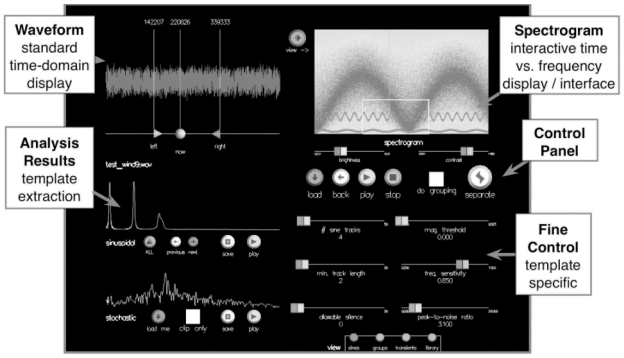

Figure 8. TAPESTREA analysis GUI.

The internal representation of a stochastic background template begins with a link to a sound file containing the related background component extracted in the analysis phase. However, merely looping through this sound file or randomly mixing segments of it does not produce a satisfactory background sound. Instead, our goal here is to generate ongoing din that sounds controllably similar to the original extracted stochastic background. Therefore, the stochastic background is synthesized from the saved sound file using an extension of the wavelet-tree learning algorithm [2]. In the original algorithm, the saved background is decomposed into a wavelet tree where each node represents a coefficient, with depth corresponding to resolution. The wavelet coefficients are computed using the Daubechies wavelet with 5 vanishing moments. A new wavelet tree is then constructed, with each node selected based on the similarity of its ancestors and first k predecessors to corresponding sequences of nodes in the original tree. The learning algorithm also takes into account the amount of randomness desired. Finally, the new wavelet tree undergoes an inverse wavelet transform to provide the synthesized time-domain samples. This learning technique works best with the separated stochastic background as input, where the sinusoidal and transient events have been removed.

This chopping and randomized re-use is somewhat similar to "granular" synthesis from computer music [19][20][21], but here the tree and derived statistics provide a specific and automatic means for structuring the sound for transformation and re-synthesis. Also, rather than chopping up the original waveform, the wavelets perform the chopping in multiple frequency bands.

TAPESTREA uses a modified and optimized version of the wavelet-tree algorithm, which follows the same basic steps but varies in details. For instance, the modified algorithm includes the option of incorporating randomness into the first level of learning, and also considers k as dependent on node depth rather than being constant. More importantly, it optionally avoids learning the coefficients at the highest resolutions. These resolutions roughly correspond to high frequencies, and randomness at these levels does not significantly alter the results, while the learning involved takes the most time. Optionally stopping the learning at a lower level thus optimizes the algorithm and allows it to run in real-time.

Further, TAPESTREA offers interactive control over the learning parameters in the form of "randomness" and "similarity" parameters. The size of a sound segment to be analyzed as one unit can also be controlled, and results in a "smooth" synthesized background for larger sizes versus a more "chunky" background for smaller sizes.

Other means for creating din in TAPESTREA involves the use of loops of single deterministic and/or transient templates with full control over randomization of pitch, timing, and frequency/regularity of occurrence. We also offer "mixed bags," which allow the synthesis of a collection of templates, selected at random and synthesized with full control over randomization of pitch, time, frequency/regularity of occurrence, etc. Figure 9 shows the control screen for synthesis, including a timeline for placing synthesized objects (top), a collection of extracted templates of various types (lower left), and controls for transformation/resynthesis of templates (lower right).

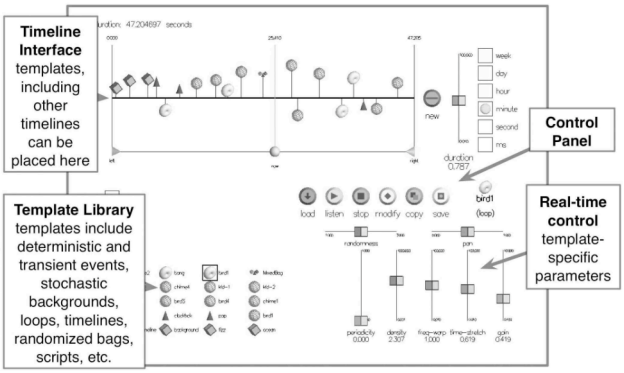

Figure 9. TAPESTREA synthesis GUI.

6. SYNTHESIS OF DIN USING CHUCK

ChucK is a new real-time audio programming language that allows precise control over timing and concurrency [22]. Similar to C++, Java, and other object-oriented languages, Chuck differs significantly by use of the ChucK operator (=>) for assignment, patching of unit generators, and other functions. Further, along with containing functions such as Std.Math (greatly extended beyond the ANSI standard C math.h library), and built-in unit generators such as SinOsc(), adc, dac, etc., all STK instrument and effects objects are compiled into ChucK as native unit generators. ChucK also provides full support for MIDI, Open Sound Control (OSC), and a variety of input devices (mice, joysticks, ASCII keyboards, Bluetooth devices (such as the Nintendo Wii controller), and the accelerometers, microphones, and cameras built into many modern laptops).

This code example shows the synthesis of applause using only one recorded soundfile as a source. The file clap.wav is loaded into 10 sound player objects (SndBuf claps[10]), and connected to a mixer (Gain object g), through a reverberator object (JCRev r) to the output sound hardware (dac). The claps[] instances each load the same sound file (clap.wav), connect to the mixer, and then "spork" (fork) a "shred" (thread) which claps forever with pseudo-randomized pitches, gains, and clapping periods. The program ends with an infinite loop that keeps the clapping going forever (until Control-C is pressed, or the "remove shred" button is pressed in the MiniAudicle [23] GUI.

// ANSII C Code example for applause synthesis in ChucK

```
Gain g => JCRev r => dac;          // gain into Reverb into Audio Out
0.1 => r.mix;                      // amount of reverb

SndBuf claps[10];                  // make 10 wave players
[0.25,0.35,0.25,0.35,0.25,0.4,0.15,0.25,0.2,0.277] @=> float rates[];
[1.0,0.75,0.8,0.85,0.9,0.95,0.7,1.05,1.1,1.15] @=> float pitches[];

int i;                             // iterator variable

for (0 => i; i < 10 ; i++)     {   // run through all 10 clappers
   "clap.wav" => claps[i].read;    // load the sound file
    claps[i] => g;                 // connect them to the mixer
   spork ~ clapper(i);             // and tell them to start clapping
}

fun void clapper(int i)     {
   while (1)   {                   // clap forever
       Std.rand2f(0.5, 1.0) => claps[i].gain;                  // random gain
       pitches[i] * Std.rand2f(0.85,1.15) => claps[i].rate; // rand. pitch
      0 => claps[i].pos;                                       // trigger wave
      rates[i] * Std.rand2f(0.9,1.1) :: second => now;      // rand. period
   }
}

while (1) 1.0 :: second => now;    // run forever so shreds stay alive
```

// END CODE EXAMPLE

Within TAPESTREA, even finer control over the synthesis can be obtained through the use of ChucK as a score/control/scripting language, used for specifying precise parameter values and for controlling exactly how these values change over time. ChucK is woven directly into the TAPESTREA synthesis GUI, and can be used to move multiple controls at a time at arbitrary rates (can't do this with a mouse!). Since ChucK allows the user to specify events and actions precisely and concurrently in time, it is straightforward to write scores to dynamically and interactively evolve a sound tapestry.

A ChucK virtual machine is attached to TAPESTREA, which registers a set of API bindings with which ChucK programs can access and control sound templates and automate tasks. Each script (called a shred) can be loaded as a sound template and be played or put on timelines. Scripts can run in parallel, synchronized to each other while controlling different parts of the synthesis. Also, scripting is an easy way to add "traditional" sound synthesis algorithms and real-time control via MIDI and Open Sound Control.

7. ADDITIONAL FILES

STK is available at:
http://ccrma.stanford.edu/software/stk/

ClapLab is available at :
http://soundlab.cs.princeton.edu/software

ClaPD and Levi Peltola's thesis are available at:
http://www.acoustics.hut.fi/publications/files/theses/lpeltola_mst

TAPESTREA is available at:
http://taps.cs.princeton.edu

ChucK and miniAudicle are available at:
http://chuck.cs.princeton.edu

8. CONCLUSIONS

This paper has described a series of related projects in the analysis and synthesis of stochastic sounds in general. Many environmental sounds are of this type, where an ensemble of individual sound-producing objects or entities combine to make a whole. The sources of such sounds can be as varied as human applause, flocks of birds, swarms of bees or locusts, wind through a forest, choirs of singing voices, and many other "crowd scenes." The author is currently assembling and editing a book with the working title: "Sonik Flox: Analysis and Synthesis of Horde Sounds," which will include some classic papers as well as new work in the field. There is still much work to be done, however, and I look forward to new advances in this challenging, often overlooked and deemphasized (many people simply resort to the use of sample loops for background sound) yet very important, area of sound analysis/synthesis.

9. #ACKNOWLEDGEMENTS

Thanks to Gary Scavone for taking co-ownership of, updating, maintaining, and documenting STK since 1998. Thanks to Steve Lakatos for using particle models for many psychoacoustic experiments. Thanks to Ge Wang for creating ChucK (with help from other Princeton soundlab members). Thanks to Spencer Salazar for creating the miniAudicle (along with Ge and others). Thanks to Ananya Misra for creating TAPESTREA (with help from other soundlab members). Thanks to Leevi Peltola, Cumhur Erkut, and Vesa Välimäki for creating ClaPD, and inviting me to co-author an IEEE paper on applause analysis/ synthesis.

10. REFERENCES

[1] Zhu X. and L. Wyse, "Sound Texture Modeling and Time-Frequency LPC," in Proc. 7th Intl. Conference on Digital Audio Effects (DAFX), Naples, Italy, 2004.

[2] Dubnov, S., Z. Bar-Joseph, R. El-Yaniv, D. Lischinski, and M. Werman, "Synthesizing sound textures through wavelet tree learning," *IEEE Computer Graphics and Applications*, 22(4), 2002.

[3] P. Cook, "Physically Informed Sonic Modeling (PhISM): Percussive Synthesis," Proceedings of the International Computer Music Conference, Hong Kong, Sept. 1996.

[4] P. Cook, "Physically Informed Sonic Modeling (PhISM): Synthesis of Percussive Sounds," Computer Music Journal, 21:3, 1997.

[5] P. Cook, "Toward Physically-Informed Parametric Synthesis of Sound Effects," Invited Keynote Address, IEEE Workshop on Applications of Signal Processing to Audio and Acoustics, October, 1999.

[6] P. Cook, "Physically Informed Stochastic Modal Sound Synthesis," Invited paper presentation at the 141st meeting of the Acoustical Society of America, Chicago, June 2001.

[7] P. Cook and G. Scavone, "The Synthesis ToolKit (STK)," Proceedings of the International Computer Music Conference, Beijing, October, 1999.

[8] G. Scavone and P. Cook, "Synthesis Toolkit in C++ (STK)," *Audio Anecdotes, Volume 2*, K. Greenebaum and R. Barzel Eds., A.K. Peters Press, 2004.

[9] S. Lakatos, P. Cook, and G. Scavone, "Selective Attention to the Parameters of a Physically Informed Sonic Model," Acoustics Research Letters Online, Journal of the Acoustical Society of America, May 2000.

[10] G. Scavone, S. Lakatos, and P. Cook, "Knowledge acquisition by listeners in a source learning task using physical models," (Invited) 139th meeting of the Acoustical Society of America, Atlanta, June, 2000.

[11] P. Cook, "Modeling Bill's Gait: Analysis and Parametric Synthesis of Walking Sounds," Proceedings of the Audio Engineering Society 22nd Conference on Virtual, Synthetic and Entertainment Audio, Helsinki, Finland, June 2002.

[12] L. Peltola, "Analysis, Parametric Synthesis, and Control of Hand Clapping Sounds," Master's Thesis, Helsinki University of Technology, 2006.

[13] L. Peltola, C. Erkut, P. Cook and V. Välimäki, "Synthesis of Hand Clapping Sounds," IEEE Transactions on Speech, Audio, and Language Processing, vol. 15, March 2007.

[14] A. Misra, P. Cook, and G. Wang, "A New Paradigm for Sound Design," Proceedings of the International Conference on Digital Audio Effects (DAFX), Montreal 2006.

[15] A. Misra, P. Cook, and G. Wang, "TAPESTREA: Sound Scene Modeling by Example," Technical Sketch, SIGGRAPH, the ACM Conference on Graphics and Interactive Technologies, Boston, 2006.

[16] A. Misra, "Musical Tapestry: Re-Composing Natural Sounds," Proceedings of the International Computer Music Conference," Winner, Journal of New Music Research Distinguished Paper Award, New Orleans, 2006.

[17] X. Serra, "A System for Sound Analysis/Transformation/ Synthesis based on a Deterministic plus Stochastic Decomposition. PhD thesis, Stanford University, 1989.

[18] T. Verma and T. Meng. "An analysis/synthesis tool for transient signals that allows a flexible sines+transients+ noise model for audio," Proceedings of 1998 IEEE International Conference on Acoustics, Speech, and Signal Processing, 1989.

[19] B. Truax, "Composing with real-time granular sound," Perspectives of New Music 28(2), 1990.

[20] B. Truax, "Genres and techniques of soundscape composition as developed at Simon Fraser University," Organised Sound 7(1), 2002.

[21] C. Roads, *Microsound*. Cambridge: MIT Press, 2002.

[22] G. Wang and P. Cook, "ChucK: A Concurrent, On-the-fly, Audio Programming Language," Proceedings of the International Computer Conference, Winner, Best Presentation Award, Singapore, Oct. 2003.

[23] S. Salazar, G. Wang, and P. Cook, "miniAudicle and the ChucK Shell: New Interfaces for ChucK Development and Performance," Proceedings of the Intlernational Computer Music Conference, New Orleans, 2006.